

Reinforcement Learning

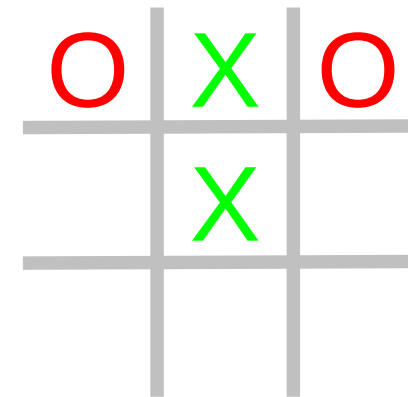
- Introduction
 - MENACE (Michie 1963)
- Formalization
 - Policies
 - Value Function
 - Q-Function
- Model-based Reinforcement Learning
 - Policy Iteration
 - Value Iteration
- Model-free Reinforcement Learning
 - Q-Learning
 - extensions
- Application Examples

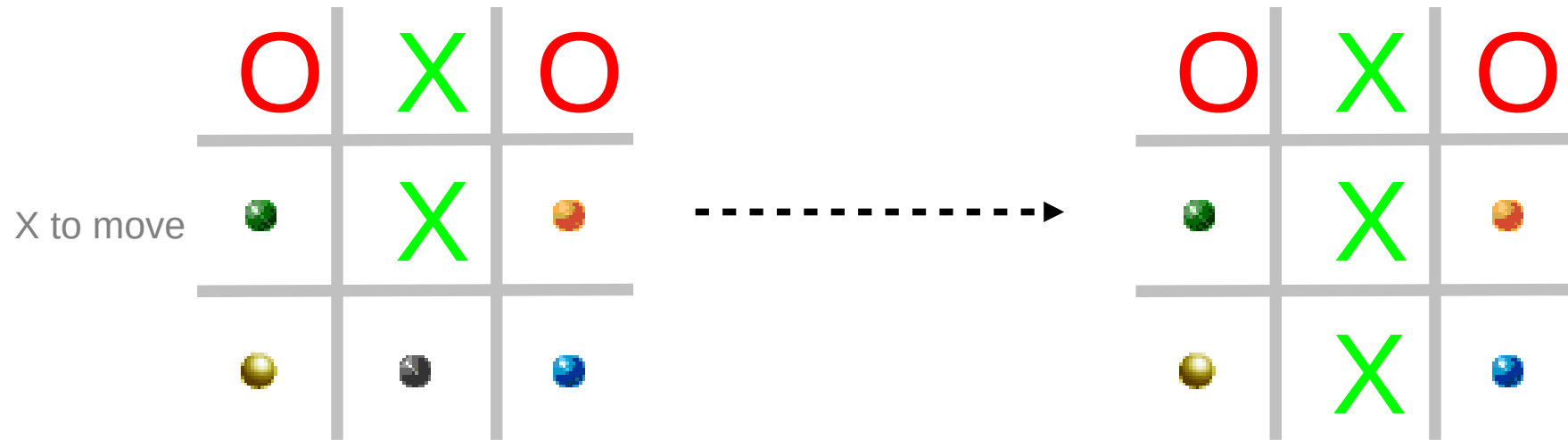
Reinforcement Learning

- **Goal**
 - Learning of policies (action selection strategies) based on feedback from the environment (reinforcement)
 - e.g., game won / game lost
- **Applications**
 - **Games**
 - Tic-Tac-Toe: MENACE (Michie 1963)
 - Backgammon: TD-Gammon (Tesauro 1995)
 - Schach: KnightCap (Baxter et al. 2000)
 - **Other**
 - Elevator Dispatching
 - Robot Control
 - Job-Shop Scheduling

MENACE (Michie, 1963)

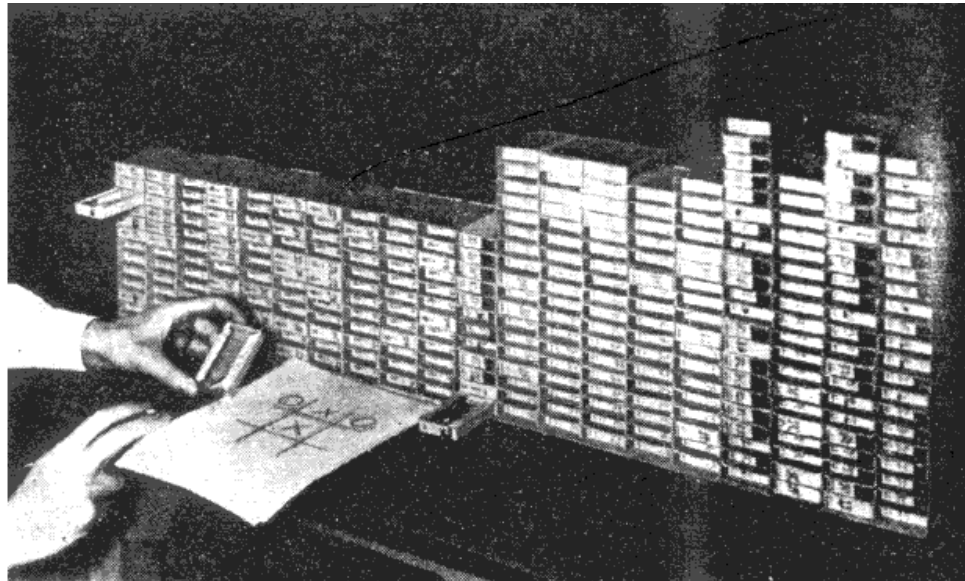
- Learns to play Tic-Tac-Toe
- Hardware:
 - 287 Matchboxes (1 for each position)
 - Beads in 9 different colors (1 color for each square)
- Playing algorithm:
 - Select the matchbox corresponding to the current position
 - Randomly draw a bead from this matchbox
 - Play the move corresponding to the color of the drawn bead
- Implementation: <http://www.codeproject.com/KB/cpp/ccross.aspx>





X to move

Select the matchbox
corresponding to
this position



Play the move that
corresponds to the
color of the drawn
bead



Draw a bead from the
matchbox



Reinforcement Learning in MENACE

- Initialisation
 - all moves are equally likely, i.e. every box contains an equal number of beads for each possible move / color
- Learning algorithm:
 - Game **lost** → drawn beads are kept (*negative reinforcement*)
 - Game **won** → put the drawn bead back and add another one in the same color to this box (*positive reinforcement*)
 - Game **drawn** → drawn beads are put back (no change)
- This results in
 - Increased likelihood that a successful move will be tried again
 - Decreased likelihood that an unsuccessful move will be repeated

Credit Assignment Problem

- Delayed Reward
 - The learner knows whether it has won or lost not before the end of the game
 - The learner does not know which move(s) are responsible for the win / loss
 - a crucial mistake may already have happened early in the game, and the remaining moves were not so bad (or vice versa)
- Solution in Reinforcement Learning:
 - All moves of the game are rewarded or penalized (adding or removing beads from a box)
 - Over many games, this procedure will converge
 - bad moves will rarely receive a positive feedback
 - good moves will be more likely to be positively reinforced

Reinforcement Learning - Formalization

- Learning Scenario
 - $s \in S$: state space
 - $a \in A$: action space
 - $s_0 \in S_0$: initial states
 - a state transition function $\delta : S \times A \rightarrow S$
 - a reward function $r : S \times A \rightarrow \mathbb{R}$
- Markov property
 - rewards and state transitions only depend on last state
 - not on how you got into this state
- State and action space can be
 - Discrete: S and/or A is a set
 - Continuous: S and/or A are infinite (not part of this lecture!)
- State transition function can be
 - Stochastic: Next state is drawn according to $\delta(s'|s, a)$
 - Deterministic: Next state is fixed $\delta(s, a) = s'$

Reinforcement Learning - Formalization

- Environment:
 - the agent repeatedly chooses an action according to some *policy* $\pi(a|s)$ *or* $\pi(s) = a$
 - this will put the agent in state s into a new state s' according to
 - stochastic: $\Pr^\pi(s'|s) = \delta(s'|s, a)\pi(a|s)$
 - deterministic: $s' = \delta(s, \pi(s))$
 - in some states the agent receives feedback from the environment (*reinforcement*)

MENACE - Formalization

- Framework
 - states = matchboxes, discrete
 - actions = moves/beads, discrete
 - policy = prefer actions with higher number of beads, stochastic
 - reward = game won/ game lost
 - *delayed* reward: we don't know right away whether a move was good or bad+
 - transition function: choose next matchbox according to rules, deterministic
- Task
 - Find a policy that maximizes the sum of future rewards

More Terminology

- **delayed reward**
 - reward for actions may not come immediately (e.g., game playing)
 - modeled as: every state s_i gives a reward r_i , but most $r_i=0$
- **trajectory:**
 - sequence of state-actions $\langle s_0, a_0, s_1, \dots, a_{n-1}, s_n \rangle$
 - A deterministic policy and transition function create a unique trajectory, stochastic policies or transition functions may result in different trajectories

Learning Task

Learning goal:

- maximize **cumulative reward** (**return**) for the **trajectories** a policy is generating
- reward from "now" until the end of time

$$R(\pi) = R(\tau^\pi) = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$$

- immediate rewards are weighted higher, rewards further in the future are discounted (**discount factor** γ)
- if not discounted, the sum to infinity could be infinite

Learning Task

- How can we compute $R(\tau^\pi)$?

$$\begin{aligned}R(\tau^\pi) &= \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \\ &= r(s_0, a_0) + \gamma r(s_1, a_1) + \gamma^2 r(s_2, a_2) \cdots \\ &= r(s_0, \pi(s_0)) + \sum_{t=1}^{\infty} \gamma^t r(\delta(s_{t-1}, \pi(s_{t-1})), \pi(s_t)) \\ &= V^\pi(s_0)\end{aligned}$$

- Sum the observed rewards (with decay)
- Value function** = return when starting in state s and following policy π afterwards

Optimal Policies and Value Functions

- Optimal policy

- the policy with the **highest expected value** for all states s

$$\begin{aligned}\pi^*(s) &= \arg \max_{\pi} V^{\pi}(s) \\ &= \arg \max_{a \in A} r(s, a) + \gamma V^{\pi^*}(\delta(s, a))\end{aligned}$$

- Always select the action that maximizes the value function for the next step, when following the optimal policy afterwards
- But we don't know the optimal policy...

Policy Iteration

- Policy Improvement Theorem
 - if it is true that selecting the first action in each state according to a policy π' and continuing with policy π is better than always following π then π' is a better policy than π
- Policy Improvement
 - always select the action that maximizes the value function of the current policy $\pi'(s) = \arg \max_{a \in A} r(s, a) + \gamma V^\pi(\delta(s, a))$
- Policy Evaluation
 - Compute the value function for the new policy
- Policy Iteration
 - Interleave steps of policy evaluation with policy improvement

$$\pi^0(s) \rightarrow V^{\pi^0}(s) \rightarrow \pi^1(s) \rightarrow \dots \rightarrow \pi^*(s)$$

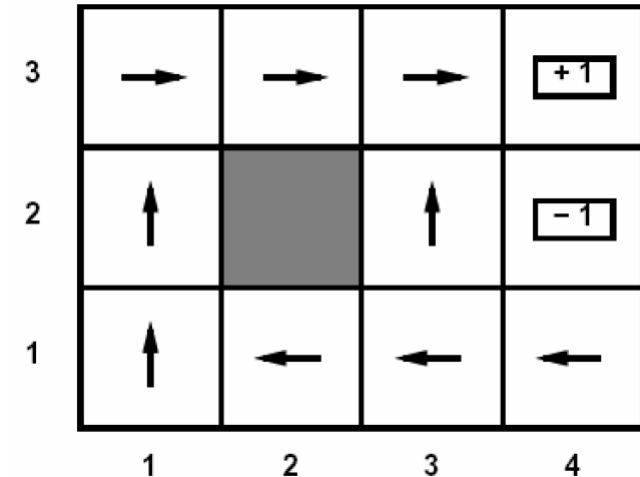
Policy Evaluation

- We need the value of all states, but can only start in s_0
 - Update all states along the trajectory
- We assumed the transition function to be deterministic, that is not realistic in many settings
 - Monte Carlo approximation
 - Create k samples and average

$$\begin{aligned}
 V^\pi(s_0) &= \mathbb{E}_{s_t} \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \\
 &= r(s_0, \pi(s_0)) + \sum_{t=1}^{\infty} \gamma^t \mathbb{E}_{s_t} \delta(s_t | s_{t-1}, \pi(s_{t-1})) r(s_t, \pi(s_t)) \\
 &= r(s_0, \pi(s_0)) + \frac{1}{k} \sum_{i=0}^k \sum_{t=1}^{\infty} \gamma^t r(s_t^i, \pi(s_t^i))
 \end{aligned}$$

Policy Evaluation - Example

- Simplified task
 - we don't know δ
 - we don't know r
 - but we are given a policy π
 - i.e., we have a function that gives us an action in each state
- Goal:
 - learn the value of each state
- Note:
 - here we have no choice about the actions to take
 - we just execute the policy and observe what happens

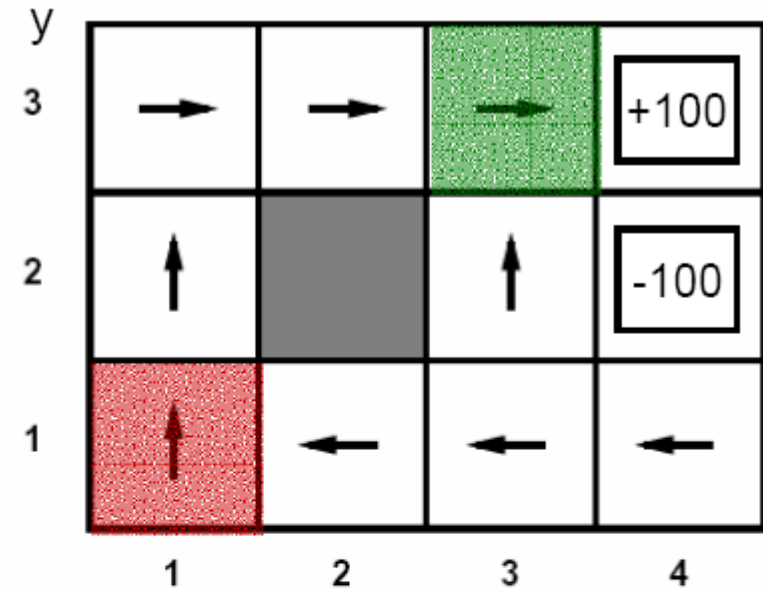


Policy Evaluation – Example

Episodes:

(1,1) up -1	(1,1) up -1
(1,2) up -1	(1,2) up -1
(1,2) up -1	(1,3) right -1
(1,3) right -1	(2,3) right -1
(2,3) right -1	(3,3) right -1
(3,3) right -1	(3,2) up -1
(3,2) up -1	(4,2) exit -100
(3,3) right -1	(done)
(4,3) exit +100	
(done)	

Transitions are
indeterministic!



$\gamma = 1,$

$$V^\pi(1,1) \leftarrow (92 + -106) / 2 = -7$$

$$V^\pi(3,3) \leftarrow (99 + 97 + -102) / 3 = 31.3$$

Policy Improvement

- Compute the value for every state
- Update the policy according to

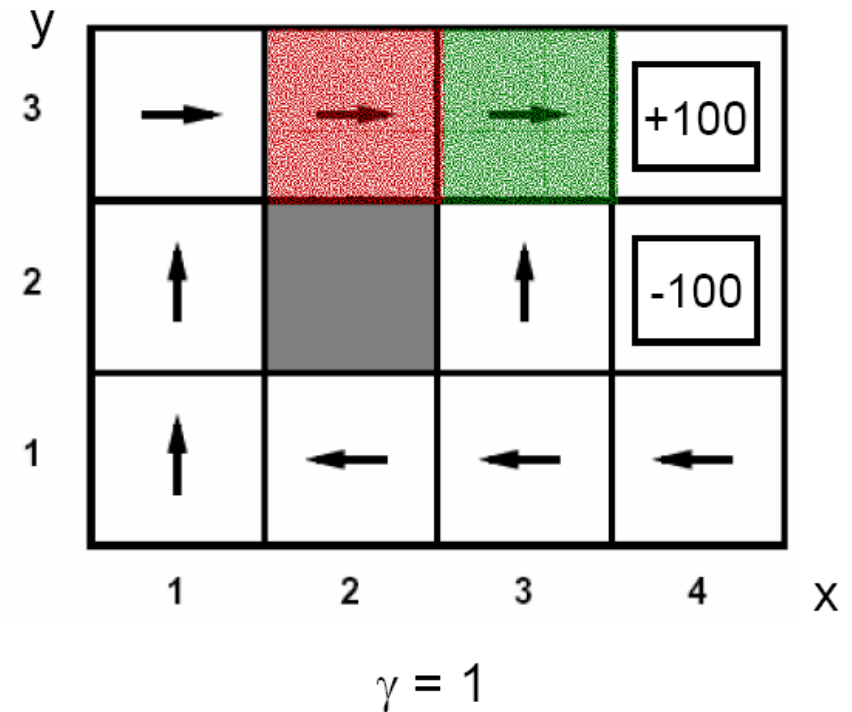
$$\pi'(s) = \arg \max_{a \in A} r(s, a) + \gamma \mathbb{E}_{s'} \delta(s'|s, a) V^\pi(s')$$

- But here we need the transition function we don't know ?

Simple Approach: Learn the Model from Data

■ Episodes:

(1,1) up -1	(1,1) up -1
(1,2) up -1	(1,2) up -1
(1,2) up -1	(1,3) right -1
(1,3) right -1	(2,3) right -1
(2,3) right -1	(3,3) right -1
(3,3) right -1	(3,2) up -1
(3,2) up -1	(4,2) exit -100
(3,3) right -1	(done)
(4,3) exit +100	
(done)	



$$\mathbf{P}((4,3) \mid (3,3), \text{right}) = 1/3$$

$$\mathbf{P}((3,3) \mid (2,3), \text{right}) = 2/2$$

But do we really need to learn the transition model?

Q-function

- the Q-function does not evaluate states, but evaluates state-action pairs
- The Q-function for a given policy π
 - is the cumulative reward for starting in s , applying action a , and, in the resulting state s' , play according to π

$$\begin{aligned}
 Q^\pi(s_0, a_0) &= r(s_0, a_0) + \sum_{t=1}^{\infty} \gamma^t \mathbb{E}_{s_t} \delta(s_t | s_{t-1}, a_{t-1}) r(s_t, \pi(s_t)) \\
 &= r(s_0, a_0) + \frac{1}{k} \sum_{i=0}^k \sum_{t=1}^{\infty} \gamma^t r(s_t^i, a_t^i) \mid s_t \sim \delta(s_t | s_{t-1}, \pi(s_{t-1}))
 \end{aligned}$$

- Now we update the policy without the transition function

$$\pi'(s) = \arg \max_a Q^\pi(s, a)$$

Exploration vs. Exploitation

- The current approach requires us to evaluate every action
 - We need to sample each state (that is reachable from s_0)
 - We need to compute argmax_a over all available actions
- Exhaustive sampling is unrealistic
 - The state/action space may be very large, even infinite (continuous)
 - We approximate an expectation, hence multiple samples for every state/action are required
- We need to decide where to sample the transition function
 - Interesting = visited by the optimal policy
 - But we don't know the optimal policy till the end

Exploration vs. Exploitation

- **Exploit**
 - Use the action we assume to be the best
 - Approximate the optimal policy
- **Explore**
 - Optimal action may be wrong due to approximation errors
 - Try a suboptimal action
- **Define probabilities for exploration and exploitation**
 - Policy evaluation with stochastic policy

$$Q^\pi(s_0, a_0) = r(s_0, a_0) + \frac{1}{k} \sum_{i=0}^k \sum_{t=1}^{\infty} \gamma^t r(s_t^i, a_t^i) \mid s_t^i \sim \text{Pr}^\pi(s_t^i \mid s_{t-1}^i)$$

- Well-defined tradeoff can substantially reduce sample counts
- Most relevant problem for reinforcement learning

Exploration vs. Exploitation

- ϵ -greedy

- Fixed probability for selecting a suboptimal action

$$\pi'(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|} & \text{if } a = \arg \max_{a \in A} Q^\pi(s, a) \\ \frac{\epsilon}{|A|} & \text{otherwise} \end{cases}$$

- Soft-Max

- Action probability related to expected value

$$\pi'(a|s) = \frac{e^{Q^\pi(s,a)/t}}{\int e^{Q^\pi(s,a)/t}}$$

- High exploration in the beginning
- Pure exploitation at the end
- Tradeoff must change over time

Drawbacks

- Policy Iteration with Monte Carlo evaluation works well in practise with small state spaces
 - Don't learn a policy for each state, but learn the policy as a function
 - Especially well suited for continuous state spaces
 - Amount of function parameters usually much smaller than the amount of states
 - Requires well defined function space→ Direct Policy Search (not part of this lecture)
- Alternative: Bootstrapping
 - Evaluate policy based on estimates
 - May induce errors
 - But requires much lower amount of samples

Optimal Q-function

- the optimal Q-function is the cumulative reward for starting in s , applying action a , and, in the resulting state s' , play optimally (derivation: deterministic policy)

$$\begin{aligned}
 Q^*(s_0, a_0) &= r(s_0, a_0) + \sum_{t=1}^{\infty} \gamma^t \mathbb{E}_{s_t} \delta(s_t | s_{t-1}, \pi^*(s_{t-1})) r(s_t, \pi^*(s_t)) \\
 &= r(s_0, a_0) + \gamma \mathbb{E}_{s_1} \delta(s_1 | s_0, a_0) r(s_1, \pi^*(s_1)) + \gamma^2 \mathbb{E}_{s_2} \delta(s_2 | s_1, \pi^*(s_1)) r(s_2, \pi^*(s_2)) + \dots \\
 &= r(s_0, a_0) + \gamma (\mathbb{E}_{s_1} \delta(s_1 | s_0, a_0) r(s_1, \pi^*(s_1)) + \gamma \mathbb{E}_{s_2} \delta(s_2 | s_1, \pi^*(s_1)) r(s_2, \pi^*(s_2)) + \dots) \\
 &= r(s_0, a_0) + \gamma \mathbb{E}_{s_1} \delta(s_1 | s_0, a_0) Q^*(s_1, \pi^*(s_1)) \\
 &= r(s_0, a_0) + \gamma \mathbb{E}_{s_1} \delta(s_1 | s_0, a_0) \max_{a_1 \in A} Q^*(s_1, a_1)
 \end{aligned}$$

- Bellman equation:** $Q(s, a) = r(s, a) + \gamma \mathbb{E}_{s'} \delta(s' | s, a) \max_{a' \in A} Q(s', a')$
 - the value of the Q-function for the current state s and an action a is the same as the sum of
 - the reward in the current state s for the chosen action a
 - the (discounted) value of the Q-function for the best action that I can play in the successor state s'

Better Approach: Directly Learning the Q-function

- Basic strategy:
 - start with some function \hat{Q} , and update it after each step
 - in MENACE: \hat{Q} returns for each box s and each action a the number of beads in the box

- **update rule:**

- the Bellman equation will in general not hold for Q i.e., the left side and the right side will be different

$$Q(s, a) = r(s, a) + \gamma \mathbb{E}_{s'} \delta(s' | s, a) \max_{a' \in A} Q(s', a')$$

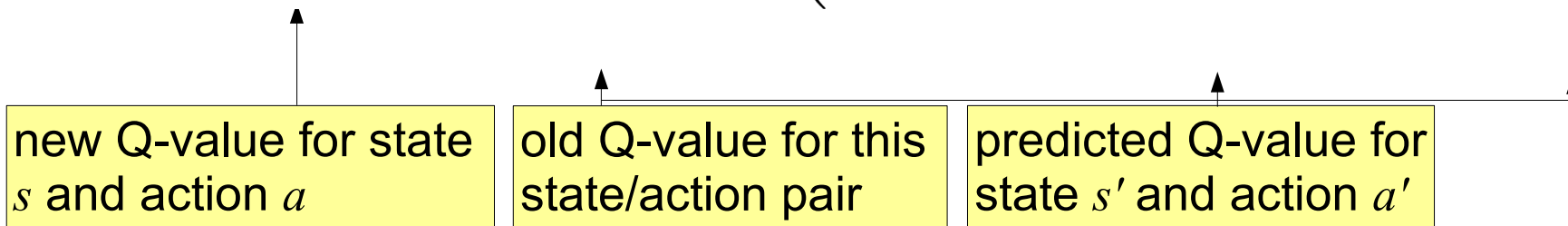
- We can not easily compute the expectation
- But we have multiple samples that contribute to the expectation

Better Approach: Directly Learning the Q-function

- Update Q-Function whenever we observe a transition s, a, r, s'
- Weighted update by a **learning rate** α

$$\hat{Q}(s, a) \leftarrow (1 - \alpha)\hat{Q}(s, a) + \alpha(r(s, a) + \gamma \max_{a' \in A} \hat{Q}(s', a'))$$

$$\leftarrow \hat{Q}(s, a) + \alpha \left(r(s, a) + \gamma \max_{a' \in A} \hat{Q}(s', a') - \hat{Q}(s, a) \right)$$



Q-learning (Watkins, 1989)

1. initialize all $\hat{Q}(s, a)$ with 0
2. observe current state s
3. loop
 1. select an action a and execute it
 2. receive the immediate reward and observe the new state s'
 3. update the table entry

$$\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha \left[\underbrace{(r(s, a) + \gamma \max_{a'} \hat{Q}(s', a'))}_{\text{red underline}} - \underbrace{\hat{Q}(s, a)}_{\text{blue underline}} \right]$$

4. $s = s'$

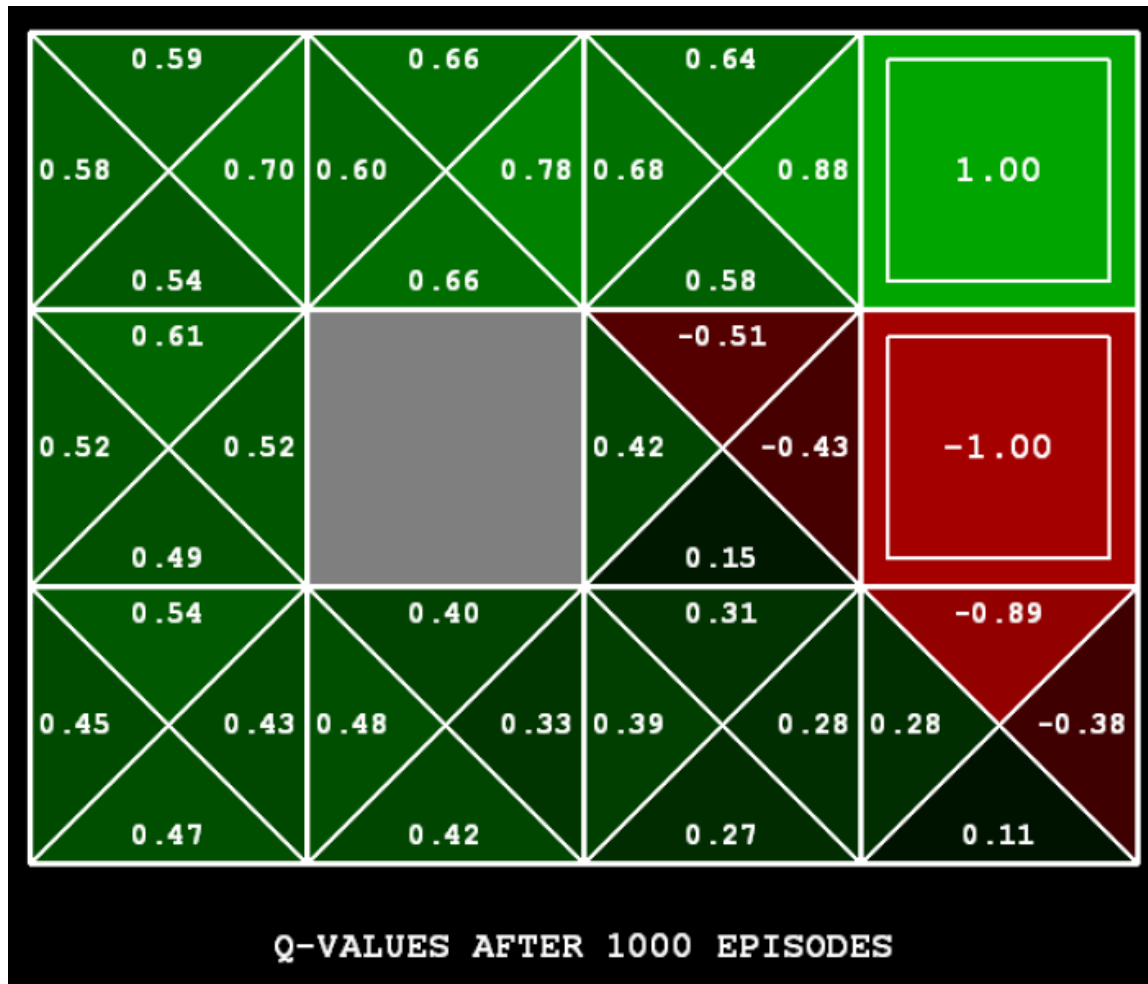
Temporal Difference:

Difference between the estimate of the value of a state/action pair **before** and **after** performing the action.

→ **Temporal Difference Learning**

Example: Maze

- Q-Learning will produce the following values



Miscellaneous

- **Weight Decay:**

- α decreases over time, e.g. $\alpha = \frac{1}{1 + \text{visits}(s, a)}$

- **Convergence:**

it can be shown that Q-learning converges

- if every state/action pair is visited infinitely often
 - not very realistic for large state/action spaces
 - but it typically converges in practice under less restricting conditions

- **Representation**

- in the simplest case, $\hat{Q}(s, a)$ is realized with a look-up table with one entry for each state/action pair
- a better idea would be to have trainable function, so that experience in some part of the space can be generalized
- special training algorithms for, e.g., neural networks exist

Drawbacks of Q-Learning

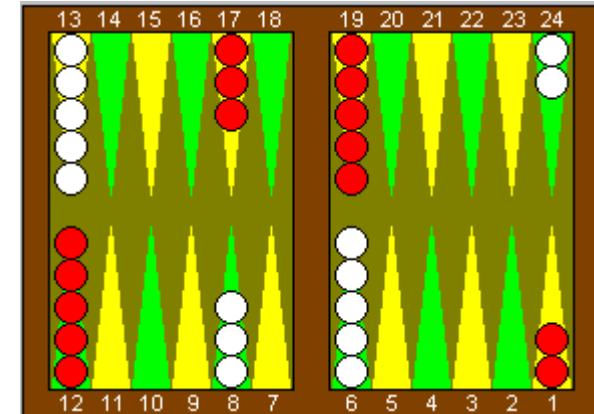
- We still need to compute $\arg \max a$, requiring estimates for all actions
 - $\arg \max a$ is the optimal policy
 - our policy converges to the optimal policy→ don't use $\arg \max a$, but the action from the current policy
- perform *on-policy updates*
 - update rule assumes action a' is chosen according to current policy
 - Update whenever observing a sample s, a, r, s', a'
$$\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha \left(r(s, a) + \gamma \hat{Q}(s', a') - \hat{Q}(s, a) \right)$$
 - convergence if the policy gradually moves towards a policy that is greedy with respect to the current Q-function→ SARSA

Properties of RL Algorithms

- Transition Function
 - Model-based: Assumed to be known or approximated
 - Model-free
- Sampling
 - On-Policy: Samples must be from the policy we want to evaluate
 - Off-Policy: Samples obtained from any policy
- Policy Evaluation
 - Value-based: Computes a state/action value function (this lecture)
 - Direct: Compute expected return for a policy
- Exploration
 - Directed: Method guides to a specific trajectory/state/action
 - Undirected: Method allows random sampling close to the expected maximum

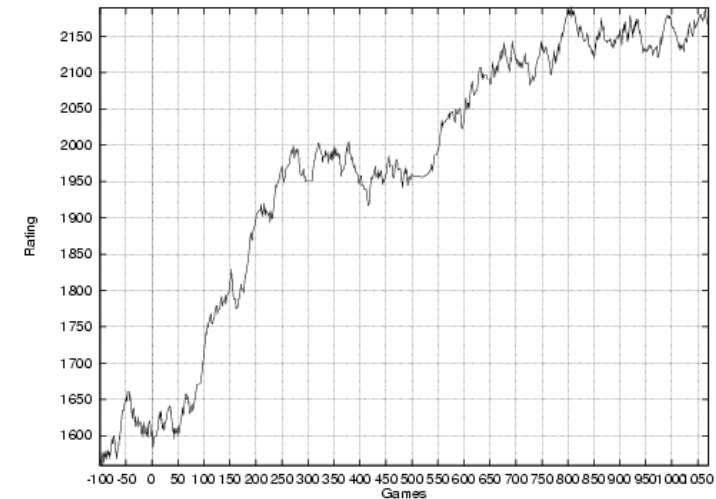
TD-Gammon (Tesauro, 1995)

- world champion-calibre backgammon program
 - Developed from beginner to world-champion strength after 1,500,000 training games against itself (!)
 - Lost World championship 1998 in a match over 100 games with a mere 8 points
 - Led to changes in backgammon theory and was used as a popular practice and analysis partner of leading human players
- Improvements over MENACE:
 - Faster convergence because of → Temporal Difference Learning
 - Neural Network instead of boxes and beads allows generalization
 - use of positional characteristics as features



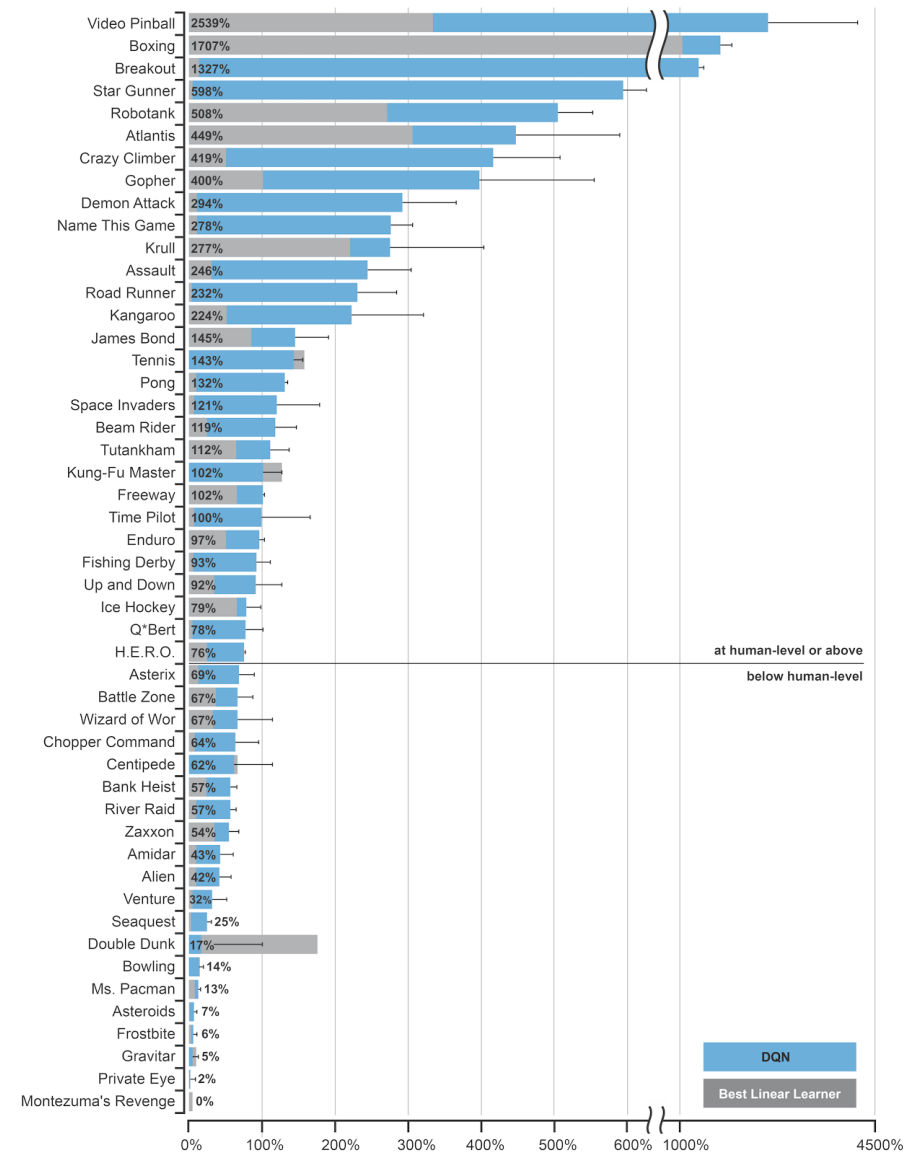
KnightCap (Baxter et al. 2000)

- Learned to play expertly in chess
 - improvement from 1650 Elo (beginner) to 2150 Elo (good club player) in only ca. 1000 games on the internet
- Improvements over TD-Gammon:
 - Integration of TD-learning with deep searches which are necessary for computer chess
 - self-play training is replaced with training by playing against various partners on the internet



Super Human ATARI playing (Minh et al. 2013)

- Reinforcement Learning with Deep Learning
- State-of-the-Art
- Better than humans in 29/49 ATARI games
- Extremely high computation times



Reinforcement Learning Resources

- Book
 - On-line Textbook on Reinforcement learning
 - <http://www.cs.ualberta.ca/~sutton/book/the-book.html>
- More Demos
 - Grid world
 - http://thierry.masson.free.fr/IA/en/qlearning_applet.htm
 - Robot learns to crawl
 - <http://www.applied-mathematics.net/qlearning/qlearning.html>
- Reinforcement Learning Repository
 - tutorial articles, applications, more demos, etc.
 - <http://www-anw.cs.umass.edu/rlr/>
- RL-Glue (Open Source RL Programming framework)
 - <http://glue.rl-community.org/>